Link-Time Attributes for LTO:
Incorporating Linker Knowledge Into the LTO Recompile

Todd Snider, Texas Instruments

2022 LLVM Developers' Meeting

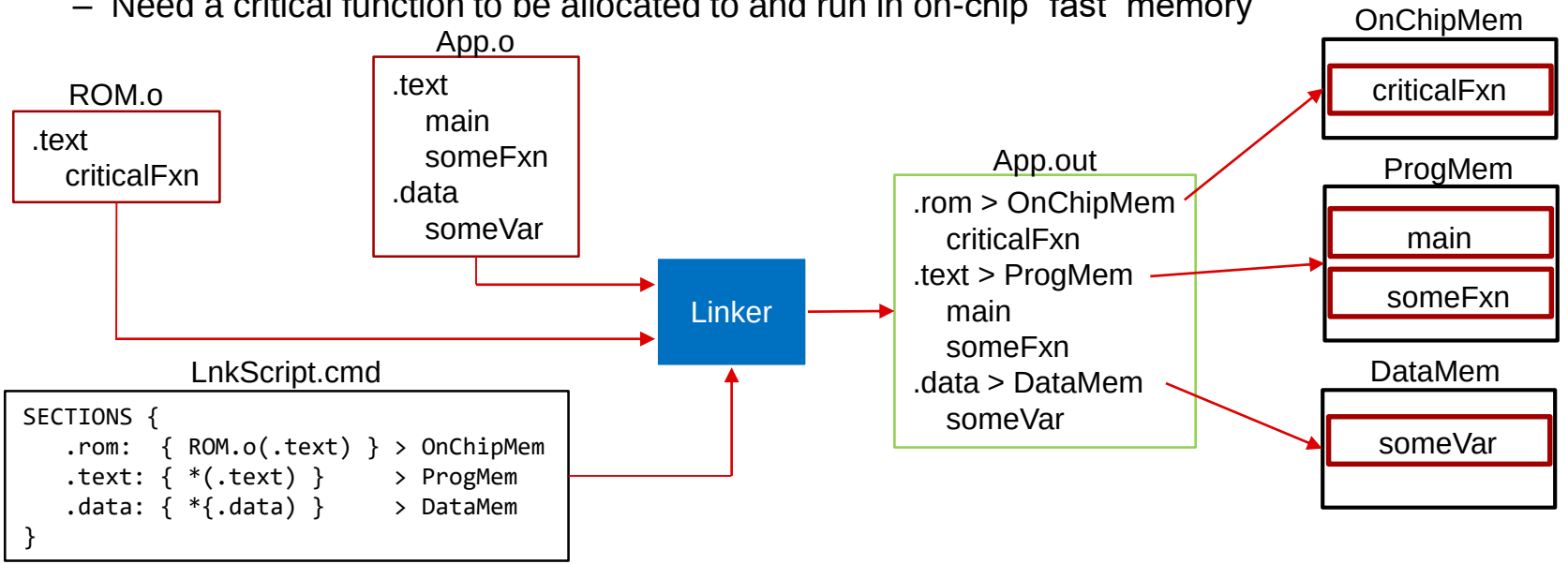
Context

- An implementation of LTO for embedded applications
  - Using a proprietary linker that plugs into the LLVM LTO infrastructure
  - Utilizing concepts from "Bringing link-time optimization to the embedded world: (Thin)LTO with Linker Scripts" (Edler Von Koch et al; LLVM Developer Meeting – Oct 2017)

- Continuing the conversation …
  - Why should LTO care about linker scripts?
  - What does Edler Von Koch prescribe?
  - Differences between our implementation and Edler Von Koch's

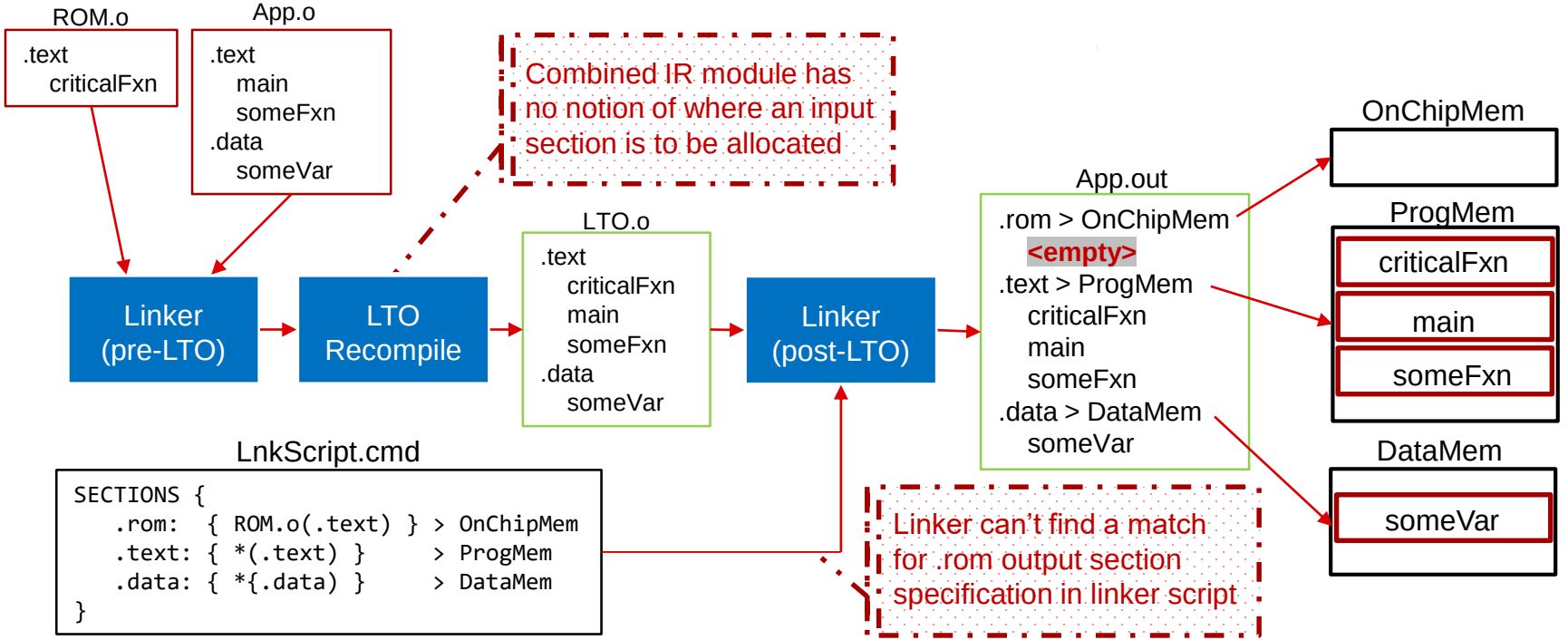
Why should LTO care about linker scripts?

- Linker scripts enable an embedded application developer to manage limited memory resources and make effective use of key system architecture features
- Adaptation/Recap of Edler Von Koch's motivational example …
  - Need a critical function to be allocated to and run in on-chip "fast" memory



Edler Von Koch's Motivational Example Revisited

- With LTO enabled …



Problem/Prescription #1

- The linker does not have the information needed to allocate an input section to its intended target memory location as indicated in the linker script

- Edler Von Koch prescribes identifying an input section's original "module ID" and passing that information to the LTO's IR linker when an IR module is merged/linked into the combined IR module

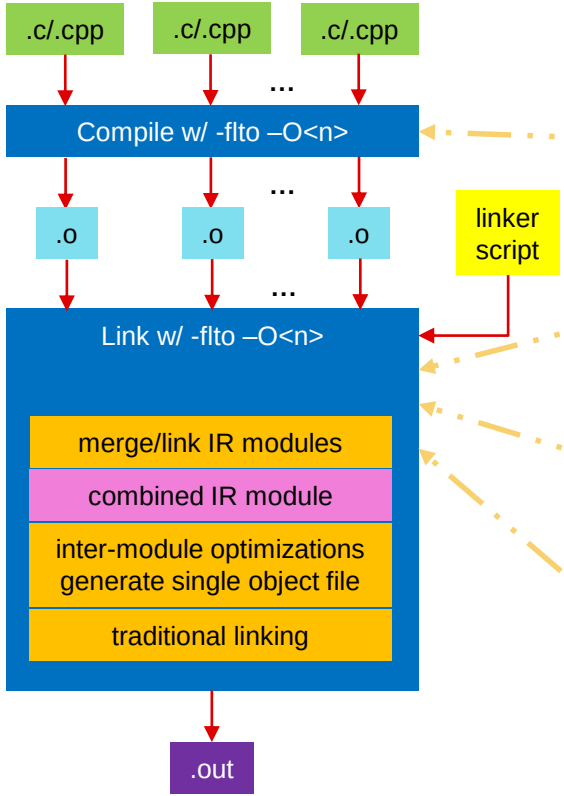
Problem #2

- Optimizations are performed across output sections during the LTO recompile since there is no awareness that symbol definitions may be allocated to different output sections during the final link
- This can cause some inter-module optimizations to be detrimental or incorrect when applied across output section boundaries
- Some examples …
  - Global Constant Merging – two constants with the same value that are allocated to separate output sections should not be merged if one of the constants is not guaranteed to be available in target memory at the time of access
  - Inlining – a function allocated to “fast” memory that is inlined into a function allocated to “slow” memory can degrade performance

Prescription #2

- Edler Von Koch prescribes that the LTO recompile needs to know the output section that an input section will be allocated into
  - Output section information should be taken into account to help determine if an inter-module optimization is safe and/or beneficial

LTO Development Flow



EVK

TI

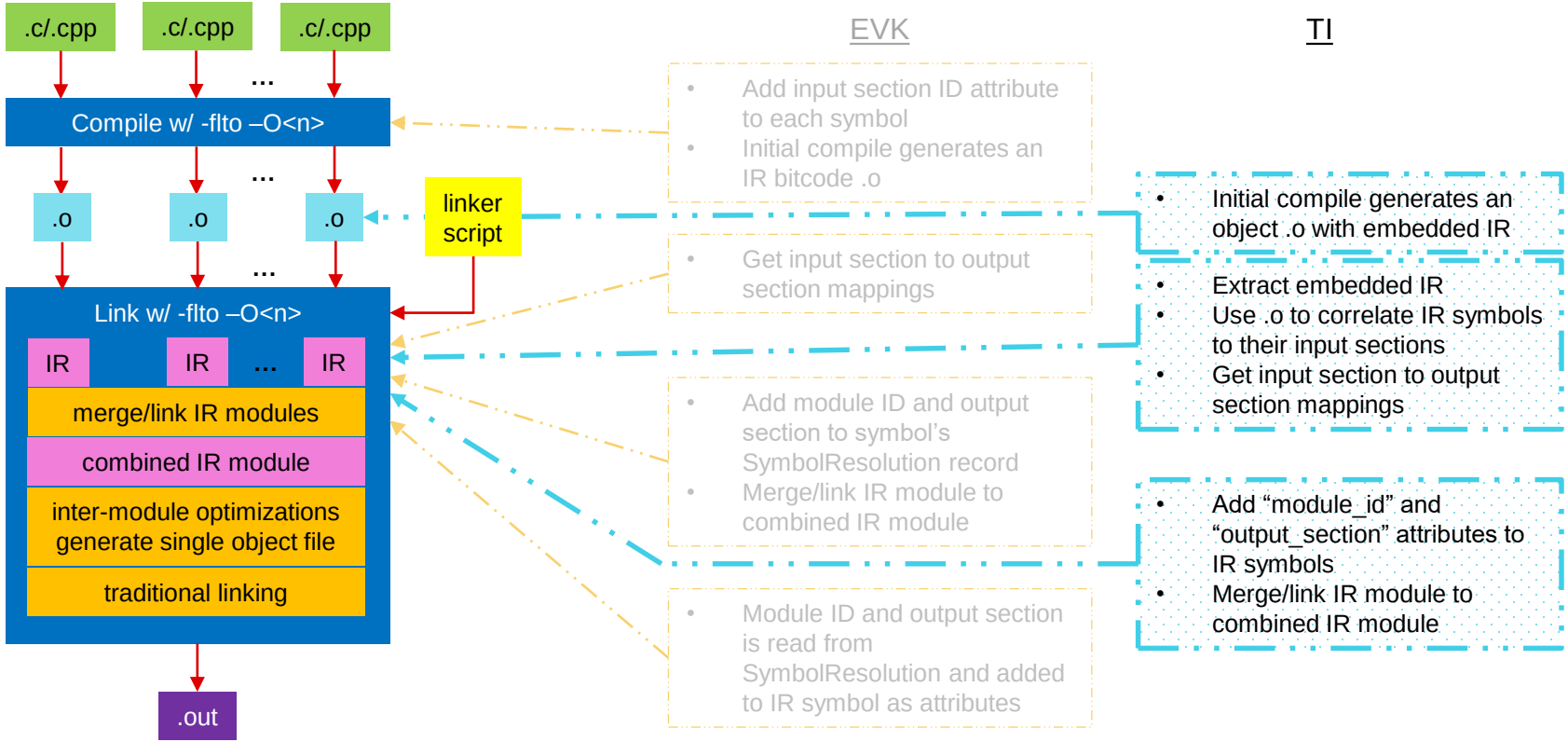
- Add input section ID attribute to each symbol
- Initial compile generates an IR bitcode .o

- Get input section to output section mappings

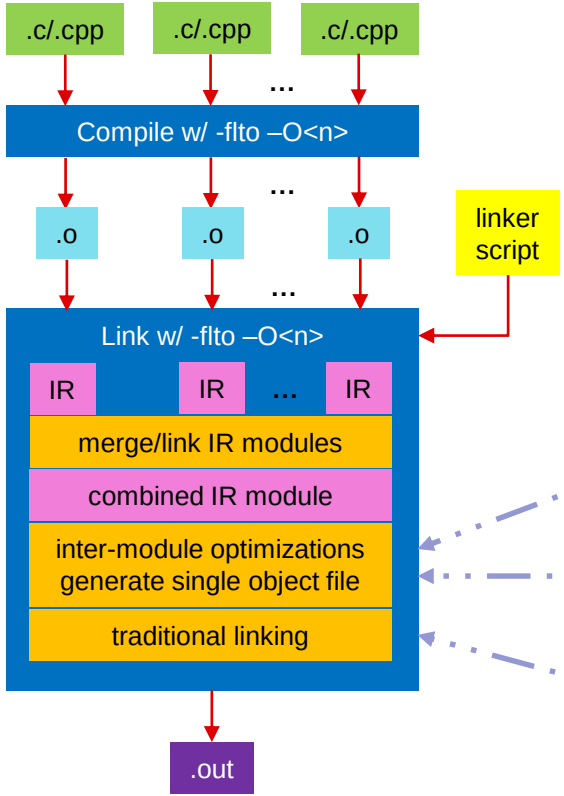
- Add module ID and output section to symbol's SymbolResolution record
- Merge/link IR module to combined IR module

- Module ID and output section is read from SymbolResolution and added to IR symbol as attributes

LTO Development Flow



LTO Recompile & Final Link



EVK & TI

- Output sections are taken into account to help decide if an optimization is safe and/or beneficial
- Symbol input section names are augmented with module ID in LTO-generated object file
- Final link uses module ID to help match input sections to their output sections

Implementation Differences

- We embed the IR module bitcode into the compiler-generated object file when `-flto` is used during the initial compile
  - An object file with embedded IR will work in the link whether LTO is enabled or not
  - Our runtime library object files have embedded IR; they can participate in LTO
  - Current upstream work is underway to support `-ffat-lto-objects` (Arda Anul)
    - We intend to adopt this in place of our current approach

Implementation Differences

- We add “output_section” and “module_id” attributes directly into an IR module before linking/merging the IR module into the combined IR module
  - Simple to implement
    - getSingleBitcodeModule() -> parseModule() -> <edit Module> -> WriteBitcodeToFile() -> LTO::add()
  - Avoids modification of SymbolResolution record
  - Utilizes existing LLVM attribute infrastructure
  - Adding additional attributes is simple

Linker Generated Symbol Attributes

- “module_id” - Original name of object file where symbol is defined
- “output_section” - Output section where definition of symbol is allocated

- Other linker knowledge that could be useful in the LTO recompile:
  - “separate_load_addr” - Whether a given output section has separate load/run placement
    - Use as a proxy for whether a symbol definition is available at the time of access
    - Help outliner decide where to define an outlined function
  - Target memory details:
    - Named memory ranges and their begin and end addresses
    - Characteristics of a named memory range (“fast”, “slow”, “shared”, etc)
    - Output section to memory range mapping
    - Maybe implement as target feature options instead of attributes?

Summary

- Takeaways
  - Our implementation is more evidence in favor of Edler von Koch's approach to accommodating linker scripts into the LTO development flow
  - Generating symbol attributes directly into IR modules vs. using SymbolResolution records to carry linker information to the LTO recompile offers advantages over Edler Von Koch's implementation:
    - No new fields needed in SymbolResolution records
    - Enables flexibility to add additional linker-knowledge to an IR module

- What's Next
  - Advocate for completion of –ffat-lto-objects support
  - Upstreaming
    - Incorporate linker-generated attributes API into lld source base
    - Incorporate awareness of linker attributes into applicable optimization passes
  - Continue work to get more useful linker-knowledge into the hands of the LTO recompile

Conversations

- Prior and Current Related Art
  - "Bringing link-time optimization to the embedded world: (Thin)LTO with Linker Scripts" (Edler Von Koch et al; LLVM Developer Meeting – Oct 2017)
  - Support for -ffat-lto-objects – Arda Anul
    - [RFC] -ffat-lto-objects support - IR & Optimizations - LLVM Discussion Forums
    - D131618 [WIP][Do NOT review] LLD related changes for -ffat-lto-objects support (llvm.org)

- Embedded Toolchains Working Group
  - LLVM Embedded Toolchains Working Group sync up - Community - LLVM Discussion Forums

- There are many ways to make use of linker-knowledge in the LTO recompile!
  - Interested? Contact me! t-snider@ti.com

Thank you!

- Acknowledgements
  - Edler Von Koch et al, Qualcomm
  - Gaurav Mitra, Meta
  - Alan Davis, Texas Instruments